OPERATION KOMOREBI

REALIZATION CALCULUS



RESEARCH LIBRARY

Certified Bayesian Inference

One-observation Gaussian updates, exact packets, and outward-rounded evidence

Visual guide

Komorebi P.

Standardized publication edition / 8 October 2026

REALIZATION CALCULUS

Reading scope

A complete worked route from one Gaussian observation to exact packet updates, a checked posterior tolerance, and outward-rounded parameter intervals. All figures are reproducible mathematical drawings or plots.

How to use this guide

This guide follows one inference computation all the way to its returned evidence. The coordinate example appears repeatedly, so that the model, residual, packet choice, gain, posterior law, rounded checker, and output intervals can be recognized as parts of the same calculation. A second example rotates the packets and starts with a singular prior. A scalar counterexample shows exactly where a tempting rounded-cache implementation breaks.

Read Sections 1–11 for the mathematical picture, Sections 12–22 for the algorithm and arithmetic contracts, and Sections 23–35 for the rotated example, the failure case, execution evidence, and release boundaries. The traditional paper supplies the detailed theorem development. The Lean companion supplies declaration names, source locations, and selected formal proofs.

The finished mathematical object is one observation update with exact rational, factor-coded inputs. The canonical program is `Komorebi.CachedPosterior.runCached`. Its iteration is exact rational arithmetic. Its final certificate and parameter representation use outward-rounded dyadic arithmetic. The returned intervals describe the Gaussian approximation associated with the returned gain.

Two tolerances have different jobs. The posterior tolerance ε bounds a Wasserstein distance between probability laws. The output tolerance δ bounds radii of intervals around parameters of the approximation. Keeping these jobs separate makes the pictures easier to read and prevents an apparent numerical guarantee from becoming a stronger claim than the theorem states.

Evidence convention

The source release preserves a successful clean Lean build and axiom audit. This publication distinguishes those archived receipts from the fresh source and Python checks performed for the paper set. A fresh Lean compilation is reported only if the pinned runtime has actually run.

Contents

1 The complete contract in one picture **3**

2 What one observation changes **4**

3 The exact posterior is the target law **5**

4 A singular prior is a supported case **6**

5 A gain defines an approximation before it is exact **7**

6 How residual energy becomes a law error **8**

7 Packets split the residual without losing it **9**

8 Squared packet energies are an exact ledger **10**

9 The coordinate model we will follow 11

10 Six selective updates, one packet at a time 12

11 The final coordinate law is close, but distinct 13

12 What the cache actually stores 14

13 Why unselected residuals remain correct 15

14 Greedy selection is a specified finite scan 16

15 The geometric envelope controls the whole run 17

16 Finite fuel is sufficient, not a forecast 18

17 Stopping slack protects the final certificate 19

18 The final checker sees the returned gain 20

19 A rejected precision is an honest part of the run 21

20 Dyadic intervals are enclosures, not rounded guesses 22

21 The returned intervals describe one named Gaussian 23

22 Why epsilon and delta cannot be exchanged 24

23 Rotated packets organize a singular model 25

24 Four rotated updates and their final law 26

25 A rounded cache can announce false convergence 27

26 The independent verifier still rejects the bad gain 28

27 The arithmetic boundary is deliberately narrow 29

28 Three solver branches should stay distinguishable 30

29 A proof-producing instance names its actual result 31

30 Valid input and available resources are different 32

31 The cache saves iteration work, not an overall victory 33

32 What the verification record establishes 34

33 The assurance map separates theorem from interface 35

34 Extensions have concrete proof obligations 36

35 Reading the complete result back from its evidence 37

1 The complete contract in one picture

The contract begins with a probabilistic model, rather than a matrix equation alone. Let the state be $X \in \mathbb{R}^d$, let the observed vector be $Y \in \mathbb{R}^p$, and write

$$X \sim \mathcal{N}(m, P), \qquad Y = HX + \eta, \qquad \eta \sim \mathcal{N}(0, R),$$

with independent state and noise. The covariance data arrive as exact factors

$$P = LL^{\mathsf{T}}, \qquad R = NN^{\mathsf{T}} + \sigma I, \qquad \sigma > 0.$$

The factors prove positive semidefiniteness of the prior and a positive noise floor. The observation matrix may be rectangular. The prior may be singular.

The solver returns an approximate gain B and a represented Gaussian law. A final independent checker connects its residual to the exact posterior law. A separate evaluator returns mean and covariance intervals for that same approximate Gaussian. These are successive mathematical obligations, not several interchangeable ways to print a result.

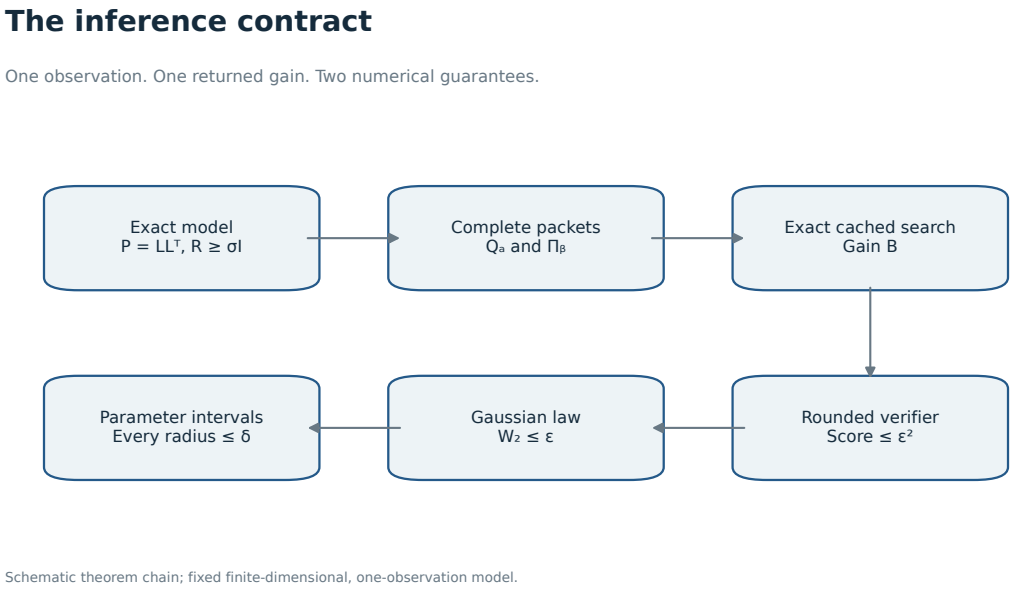


Figure 1: The inference contract connects model data, the actual returned gain, posterior-law error, and parameter enclosures. Each arrow has its own formal obligation.

The diagram is a reading map. The interval layer does not replace the posterior layer, and an internal score does not replace final verification. The cached solver reaches the contract by preserving an invariant throughout its computation and then checking the returned candidate independently.

Source route: `RELEASE_SCOPE.md`; `CachedPacketOutput.lean`; `PosteriorOutputSemantics.lean`.

2 What one observation changes

Before the observation, the model describes a distribution of possible states. After a particular value y is observed, inference describes the conditional distribution of the state. The observed value changes the posterior mean through the innovation

$$v = y - Hm.$$

It also changes the strength of the error certificate because a gain error can be amplified by a large innovation. The posterior covariance of the Gaussian linear model is determined by H, P, R and does not depend on the numerical value of y.

The innovation covariance is

$$S = HPH^{\mathsf{T}} + R.$$

The noise floor implies $S \succeq \sigma I$, so S is invertible even when P is not. This is the important asymmetry in the model. State uncertainty can live on a lower-dimensional subspace while observation noise makes the innovation equation well posed.

The guide treats y as exact mathematical data. A decimal string such as `0.1` can be interpreted as the rational $1/10$ by the reference parser, but an uncertain measurement requires a different input contract. Likewise, repeating this procedure with a new prior at every time step is a proposed sequential extension. The release theorem covers the complete computation for this one observation, including its certificate and output.

Source route: `RationalCertificateChecker.lean`; `FDInnovationConditioning.lean`; `RELEASE_SCOPE.md`.

3 The exact posterior is the target law

The exact gain and posterior parameters are

$$K = PH^{\mathsf{T}}S^{-1}, \qquad m_* = m + Kv, \qquad C_* = P - KHP.$$

The closed form gives the target Gaussian $\mathcal{N}(m_*, C_*)$. It is not merely a formal solution of the gain equation. The source proves that this Gaussian kernel is a version of conditioning for the joint Gaussian model.

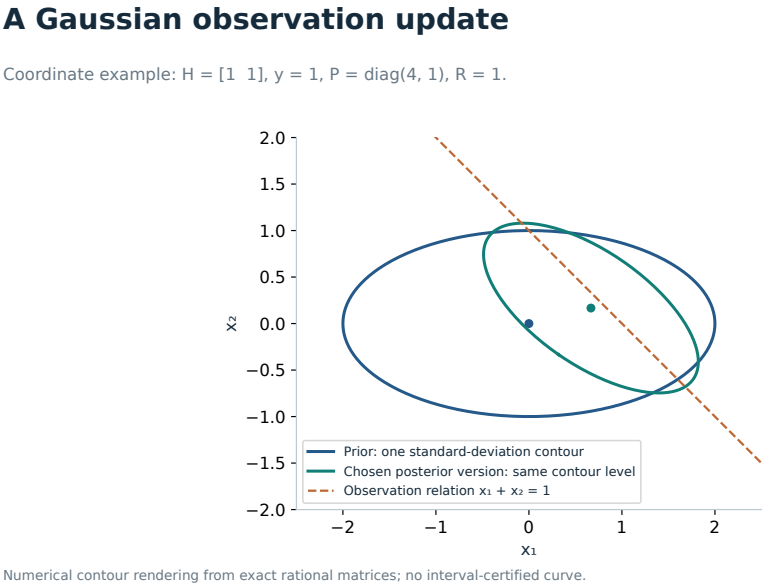


Figure 2: Numerical contours of exact coordinate prior and chosen posterior matrices. The one-standard-deviation level sets are not confidence regions. With noise variance one, the dashed observation relation is not a state-support constraint.

The posterior kernel has an explicit value at every y. Its identification with a canonical conditional law is an almost-everywhere statement with respect to the observation distribution, as conditional distributions are generally determined only almost everywhere. The explicit Gaussian version is the one used in the pointwise residual certificate.

This distinction matters because the guide later proves a bound at the supplied observation value. The mathematical target is the chosen, explicitly defined posterior version. The proof also establishes its Bayesian meaning. We do not infer pointwise equality of arbitrary versions of a conditional law from an almost-everywhere theorem.

Source route: `FDPosteriorKernel.lean`; `FDGaussianConditioning.lean`; `FDInnovationConditioning.lean`.

4 A singular prior is a supported case

A Gaussian covariance need not be invertible. In the rotated example,

$$L = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad P = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

The state is supported on the line $x_1 = x_2$. The direction $x_1 = -x_2$ has zero prior variance. Drawing a filled two-dimensional density for this law would be misleading: the probability measure is concentrated on a line.

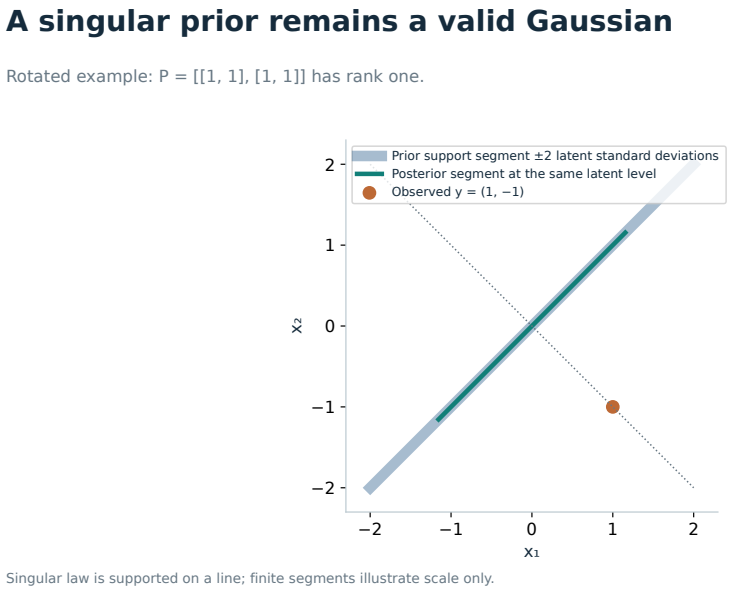


Figure 3: Prior and chosen posterior are supported on the diagonal line. The displayed finite segments mark two latent standard deviations and show only a scale within the unbounded support. Positive observation noise makes S invertible.

With $H = I$ and $R = I$, the innovation covariance becomes

$$S = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

Its eigenvalues are 3 and 1. The gain can therefore be computed and certified without taking an inverse of P. The observation $y = (1, -1)^{\mathsf{T}}$ lies in the zero-prior-variance direction. In this example it does not move the state mean, because the exact gain annihilates that direction. It still enters the residual weight through $\|y - Hm\|^2$. The probability model and the certificate therefore use the same observation in different, precisely specified ways.

Source route: `examples/adaptive_rotated.json`; `FDPosteriorCovPSD.lean`; `FDGainSampler.lean`.

5 A gain defines an approximation before it is exact

An arbitrary gain $B \in \mathbb{Q}^{d\times p}$ defines the innovation mean

$$m_B = m + Bv$$

and the Joseph covariance

$$C_B = (I - BH)P(I - BH)^{\mathsf{T}} + BRB^{\mathsf{T}}.$$

Each term is positive semidefinite. Thus C_B is a valid covariance even when B does not solve the exact gain equation. This makes every iterate a mathematically meaningful Gaussian approximation.

The solver does not need to declare the current approximation exact in order to use it. It instead measures how far its gain is from satisfying

$$BS = PH^{\mathsf{T}}.$$

The residual

$$F(B) = PH^{\mathsf{T}} - BS$$

vanishes at K. Since S is invertible, $F(B) = 0$ implies $B = K$.

For the coordinate example, the approximate covariance remains valid during the six selective updates. At the last update it is close to the exact posterior covariance but is still the Joseph covariance associated with a nonexact rational gain. The output intervals later enclose this particular covariance. They are not substituted into the probabilistic theorem as though rounded centers defined an independently certified Gaussian.

Source route: `FDInexactGainAlgebra.lean`; `FDGainSampler.lean`; `RationalPosteriorCertificate.lean`.

6 How residual energy becomes a law error

The residual certificate uses a coupling of the exact and approximate Gaussian laws. Both are constructed from the same centered Gaussian state and noise. Comparing the two constructions yields a difference whose mean and covariance can be evaluated. A coupling bounds Wasserstein distance through its expected squared displacement.

The sharper residual expression contains S^{-1}, but the positive floor supplies an inverse-free majorant. Define

$$E(F) = \sum_{i,j} F_{ij}^2, \qquad w = \frac{\sigma + \|v\|^2}{\sigma^2}, \qquad a(B) = wE(F(B)).$$

The checked implication is

$$W_2(\Pi_y, \mathcal{N}(m_B, C_B))^2 \le a(B),$$

where Π_y is the explicit posterior version. Consequently $a(B) \le \varepsilon^2$ is sufficient for a posterior tolerance ε.

The scalar $a(B)$ is a conservative certificate, rather than a claim that the Wasserstein distance equals the residual norm. The coupling need not be optimal, and the floor majorant can be loose. Its advantage is that it requires only exact model data and residual arithmetic, not an exact inverse or a numerical matrix square root. This is why the same score can guide an iterative gain computation and later feed a formally checked posterior guarantee.

Source route: `FDGainSampler.lean`; `FDInexactGainCoupling.lean`; `FDResidualEnergyCertificate.lean`.

7 Packets split the residual without losing it

Let Q_a be output-space projectors. They are fixed throughout this solver run and satisfy

$$Q_a^{\mathsf{T}} = Q_a, \qquad Q_a Q_b = 0 \ (a \neq b), \qquad Q_a^2 = Q_a, \qquad \sum_a Q_a = I.$$

These equations make the residual pieces $G_a = Q_a F(B)$ orthogonal and complete. They are directions of the gain equation's unresolved error, not separate observations.



Figure 4: The coordinate output projectors split the initial residual into orthogonal first- and second-coordinate pieces. Their exact squared energies add to the full energy. Input splitting is trivial for this one-dimensional observation space.

Input packets also occur in the inherited final certificate. The cached iteration itself selects output packets. The two families have distinct roles and should not be merged into an unspecified single partition.

A packet may contain more than one coordinate. The release does not assume packets are coordinate masks, although such masks provide the simplest example. The rotated fixture uses rational projectors onto the diagonal and anti-diagonal directions. Rational projectors allow exact packet arithmetic without requiring irrational normalized basis vectors in the input representation.

Completeness is necessary for the total cached energy to describe the whole residual. Orthogonality is necessary for changing one output packet to leave the other residual pieces unchanged. Both hypotheses become visible in the cache-preservation proof.

Source route: `RationalPacketCertificates.lean`; `AdaptivePacketInvariants.lean`; `CachedPacketState.lean`.

8 Squared packet energies are an exact ledger

For each output packet define

$$e_a = E(G_a) = \sum_{i,j} (Q_a F(B))_{ij}^2.$$

Orthogonality and completeness give the energy identity

$$E(F(B)) = \sum_a e_a.$$

This identity makes a scalar ledger sufficient to track the full residual's squared Frobenius norm. It does not require taking square roots or deciding signs of approximate quantities.

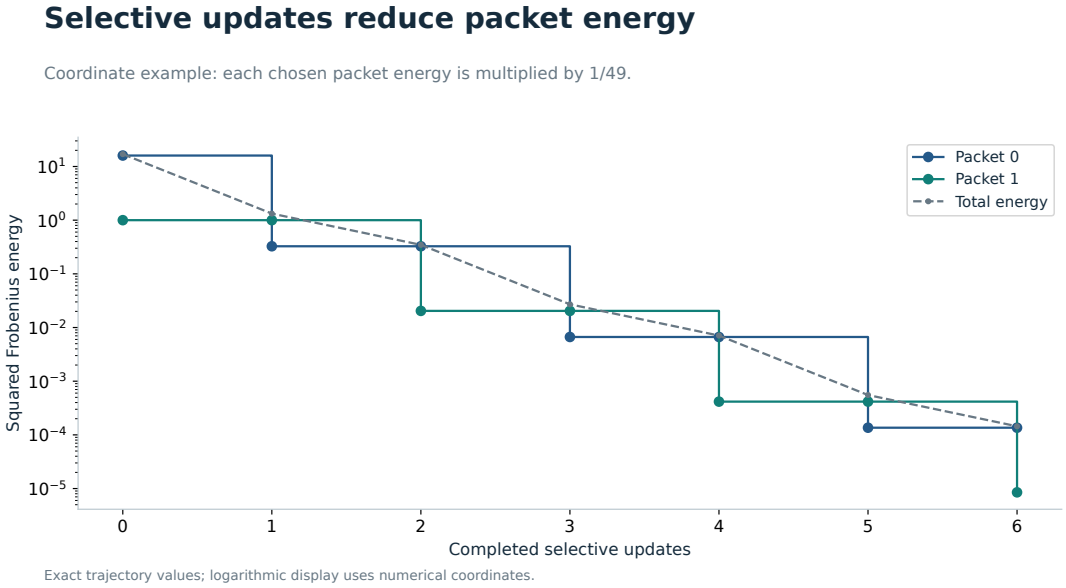


Figure 5: The coordinate energies start at 16 and 1 and decrease selectively through six updates. The final energies are $16/117649$ and $1/117649$. The logarithmic coordinates are a numerical display of these exact rational values.

The largest energy identifies the output packet carrying the largest share of the current squared residual. The greedy step acts on that packet. After the update, its energy shrinks while the unselected energies remain exactly the same. A later step may select a different packet because the ordering has changed.

These statements depend on exact rationals. Subtracting the old packet energy and adding the new one is algebraically exact here. A floating-point total can accumulate cancellation error, and a rounded residual center can lose part of the true residual. Section 25 shows why that is a mathematical boundary rather than a minor implementation detail.

Source route: `PacketSelectiveAlgebra.lean`; `CachedPacketState.lean`; `GreedyPacketSelection.lean`.

9 The coordinate model we will follow

Use a two-dimensional state and one observation:

$$H = \begin{pmatrix} 1 & 1 \end{pmatrix}, \quad L = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}, \quad N = 0, \quad \sigma = 1, \quad m = 0, \quad y = 1.$$

Therefore

$$P = \begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix}, \quad R = 1, \quad S = 6, \quad v = 1, \quad w = 2.$$

The output packets select the first and second coordinates. The input packet is the one-dimensional identity. Start with $B_0 = (0, 0)^{\mathsf{T}}$, and request

$$\varepsilon = \frac{1}{20}, \qquad \delta = \frac{1}{1000}.$$

The exact gain is $(2/3, 1/6)^{\mathsf{T}}$, and the initial residual is $(4, 1)^{\mathsf{T}}$. The gain equation is therefore simple enough to inspect by hand, while still exhibiting changing greedy packet choices, exact caching, posterior approximation, and a failed low-precision certificate.

The reference producer treats every listed integer and fraction as exact data. Its archived result records six updates, both certificate attempts, the final output arrays, and a generated Lean execution instance. The benchmark report uses a separately measured run with the same mathematical model. Timing numbers belong to that run, whereas the rational gain and residual statements are properties of the exact calculation.

Source route: `examples/adaptive_coordinate.json`; `results/cached_coordinate.json`; `GeneratedCachedCoordinate.lean`.

10 Six selective updates, one packet at a time

The step size is

$$\tau = \frac{1}{\sigma + \operatorname{tr} S} = \frac{1}{7},$$

so selecting a coordinate multiplies that coordinate's residual by $1 - \tau S = 1/7$. The gain update is $B' = B + \tau Q_a F(B)$.

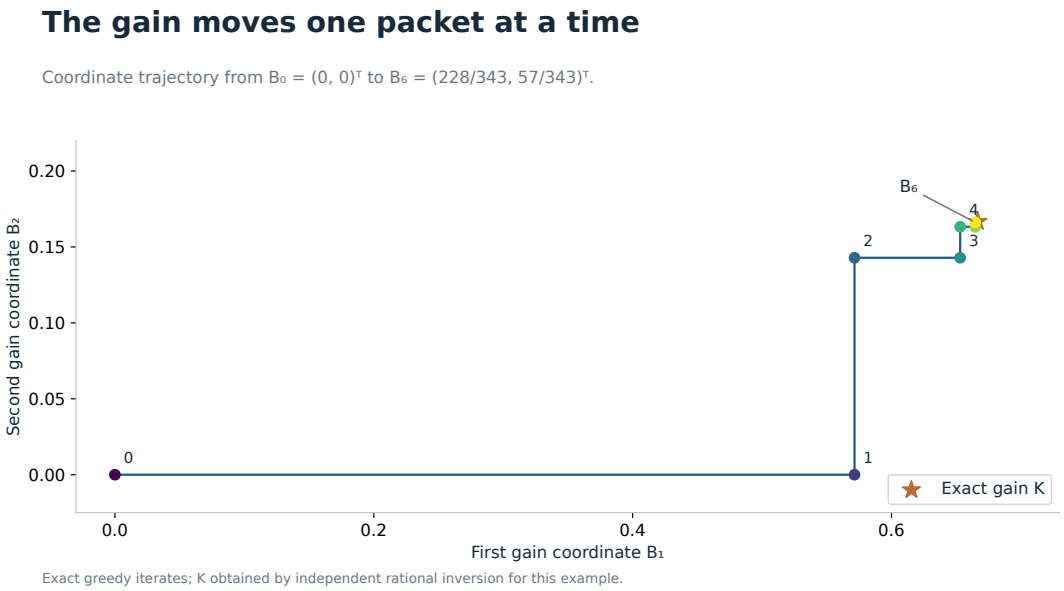


Figure 6: The coordinate run selects $0, 1, 0, 1, 0, 1$. Only the selected residual component changes at each step; after six updates the residual is $(4, 1)^{\mathsf{T}}/343$.

The exact total energies after successive updates are

$$17,\ \frac{65}{49},\ \frac{17}{49},\ \frac{65}{2401},\ \frac{17}{2401},\ \frac{65}{117649},\ \frac{17}{117649}.$$

Initially packet 0 wins because $16 > 1$. After it is reduced to $16/49$, packet 1 wins because $1 > 16/49$. The alternating pattern here follows from these specific values. The general solver does not promise alternation or a fixed cyclic schedule.

The final gain is

$$B_6 = \begin{pmatrix} 228/343 \\ 57/343 \end{pmatrix}.$$

The exact score is $34/117649$, below the search threshold $\varepsilon^2/4 = 1/1600$. At five updates the score is $130/117649$, still above that stricter threshold. This explains the sixth update without appealing to an arbitrary iteration budget.

Source route: `CachedPacketTrajectory.lean`; `GeneratedCachedCoordinate.lean`; `results/cached_coordinate.json`.

11 The final coordinate law is close, but distinct

For the exact coordinate posterior,

$$m_* = \begin{pmatrix} 2/3 \\ 1/6 \end{pmatrix}, \qquad C_* = \begin{pmatrix} 4/3 & -2/3 \\ -2/3 & 5/6 \end{pmatrix}.$$

The approximate mean is $m_{B_6} = B_6$. Its Joseph covariance has the exact identity

$$C_{B_6} = C_* + \frac{1}{6 \cdot 117649} \begin{pmatrix} 16 & 4 \\ 4 & 1 \end{pmatrix}.$$

The added matrix is positive semidefinite. This identity is an informative diagnostic, while the released certificate bounds the whole probability law through residual energy.

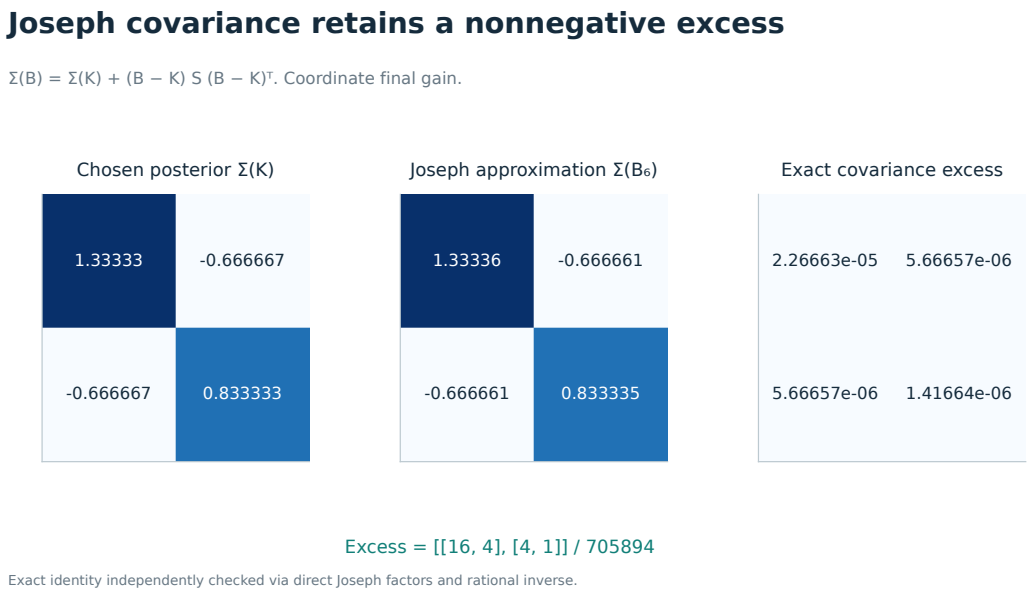


Figure 7: Decimal renderings of exact covariance entries show the Joseph excess $\left(\begin{smallmatrix} 16 & 4 \\ 4 & 1 \end{smallmatrix}\right)/705894$. This positive semidefinite rank-one matrix is an algebraic covariance difference, not the Wasserstein distance.

The laws need not be visually distinguishable at ordinary scale to remain mathematically distinct. The paper's guarantee is $W_2 \leq 1/20$, not equality of gains or distributions. A plot that overlays contours cannot replace that guarantee.

This is also why reporting only a decimal mean would discard useful evidence. The exact rational gain identifies the approximation, the final verifier establishes its posterior tolerance, and the interval output represents its parameters with declared enclosure radii. The three outputs have compatible meanings because they refer to the same returned gain.

Source route: `FDInexactGainAlgebra.lean`; `GeneratedCachedCoordinate.lean`; `PosteriorOutputSemantics.lean`.

12 What the cache actually stores

The cache state is a tuple

$$c = (B, (G_a)_a, (e_a)_a, t),$$

where B is the current gain, G_a is a projected residual matrix, e_a is its squared energy, and t is the total. The validity predicate requires

$$G_a = Q_a F(B), \qquad e_a = E(G_a), \qquad t = \sum_a e_a.$$

These equations are obligations about the stored values, rather than comments attached to a data structure.

One cache entry changes

First coordinate update: τ = 1/7 and T = 1/7.

Entry	Before	After selecting 0
Gain B	$(0, 0)^\top$	$(4/7, 0)^\top$
Residual G_0	$(4, 0)^\top$	$(4/7, 0)^\top$
Energy e_0	16	16/49
Residual G_1	$(0, 1)^\top$	$(0, 1)^\top$
Energy e_1	1	1
Total	17	65/49

Exact cache state transition. Initialization computes the full residual once.

Figure 8: A cached update refreshes one residual and one energy. The total changes by subtracting the old selected energy and adding the new one.

Initialization forms the full residual once and constructs every packet from it. Thus the initial cache has a direct proof of validity. The canonical solver computes its model constants internally through `prepare`, which supplies τ, $T = I - \tau S$, and w.

For a valid cache, $tw = a(B)$. The score then has a precise semantic meaning even though a full residual is not recomputed inside every iteration. This is the central benefit of the invariant: the algorithm may save work while preserving the quantity used by the theorem. A caller cannot obtain this meaning merely by presenting an arbitrary tuple of numbers with the same field names.

Source route: `CachedPacketState.lean`: `initialState`, `Valid`, `prepare`, `score_eq`.

13 Why unselected residuals remain correct

Suppose packet a is selected. The gain changes by $\tau Q_a F(B)$, so

$$F(B') = F(B) - \tau Q_a F(B) S.$$

For a different packet b, left multiplication gives

$$Q_b F(B') = Q_b F(B) - \tau Q_b Q_a F(B) S = Q_b F(B).$$

The vanishing cross term is exactly where output-packet orthogonality enters. For the selected packet,

$$Q_a F(B') = Q_a F(B)(I - \tau S) = G_a T.$$

The cache can therefore refresh G_a by right multiplication with T, leaving the other matrices untouched.

Squaring the new selected residual gives its new energy. The total update

$$t' = t - e_a + e'_a$$

is justified by a finite-sum identity. The source proves these equations together as `step_valid` and separately exposes unchanged residuals and energies.

Nothing in this reasoning replaces S with a measured or approximate operator. Nothing assumes an externally supplied transition matrix is correct. The semantic theorem uses the transition produced from the actual model. A changing packet family or changing model would require a fresh invariant and a transfer argument between old and new caches. That capability is outside this fixed-model run.

Source route: `AdaptivePacketInvariants.lean`; `CachedPacketState.lean`: `step_valid`, `step_other_residual`, `step_other_energy`.

14 Greedy selection is a specified finite scan

The solver selects a largest stored energy with a supplied seed and finite scan. The scan must cover every output label. Its order can be arbitrary and duplicates are allowed. When a scanned energy equals the incumbent energy, the incumbent is retained.

A finite maximum scan has deterministic ties

Scan coverage matters; ordering and duplicate visits need not be unique.

Energies	Seed	Scan	Chosen label
(1, 1)	0	[1, 0]	0
(1, 1)	1	[0, 1]	1
(16, 1)	1	[1, 0, 0]	0

Only a strict increase replaces the incumbent.

Exact selection rules; a maximum is guaranteed by complete scan coverage.

Figure 9: The maximum scan compares exact stored energies and preserves the incumbent on ties. Coverage, rather than a particular label order, is the required hypothesis.

Tie behavior is part of the executable definition. Two algorithms that both say “choose a maximum” can follow different trajectories when there are ties. The cache proof therefore establishes more than existence of some maximizing packet: it proves equality with the selection made by the recomputed greedy scan using the same seed and scan.

A missing label is a contract violation because the selected energy may no longer dominate the average of all packet energies. The seed witnesses that the output-label type is nonempty. Positive tolerance and model validity alone cannot fill either obligation.

For the coordinate and rotated examples the scan is $[0, 1]$ with seed 0. Coordinate energies yield the alternating choices already shown. The rotated example always selects 0 because the anti-diagonal residual energy is zero. Neither trace depends on approximate numerical comparisons. Approximate selection would need its own theorem relating the chosen energy to the true maximum.

Source route: `GreedyPacketSelection.lean`; `CachedPacketTrajectory.lean`: `chosen_eq`; `CachedPacketSolver.lean`.

15 The geometric envelope controls the whole run

Let $A > 0$ be the number of output labels. The contraction constants are

$$q = \left(1 - \frac{\sigma}{\sigma + \operatorname{tr} S}\right)^2, \qquad r = 1 - \frac{1-q}{A}.$$

The chosen packet has at least the average energy. Its energy contracts by at most q, so the full score obeys

$$a(B_{n+1}) \le r\, a(B_n), \qquad a(B_n) \le r^n a(B_0), \qquad 0 \le r < 1.$$

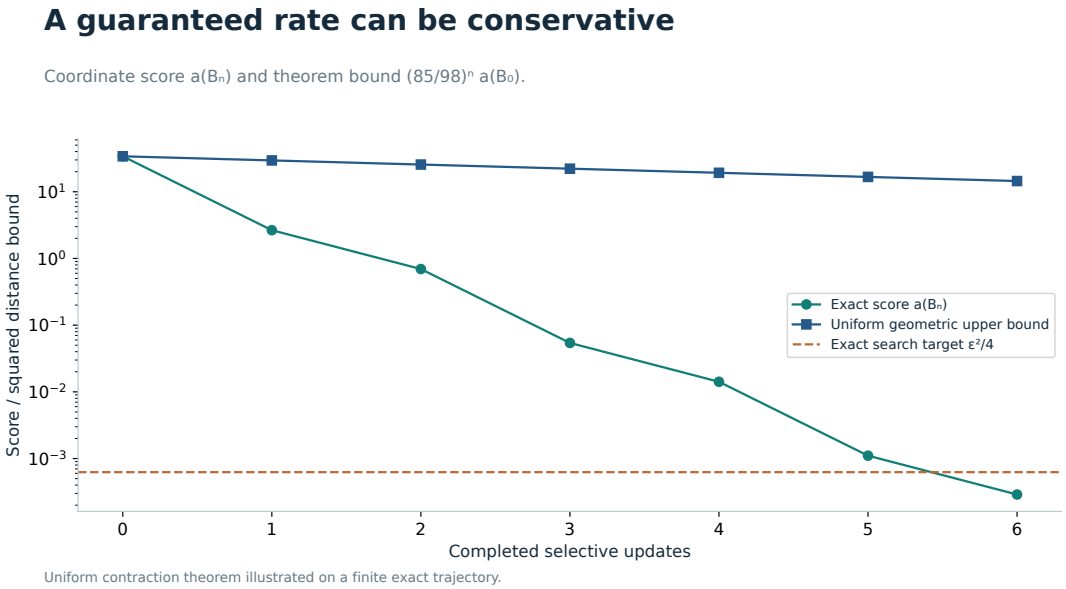


Figure 10: The realized coordinate score stays below the proven geometric envelope. The envelope supports termination and can be much looser than this particular trajectory.

For the coordinate model, $q = 36/49$ and $r = 85/98$. A selected scalar residual actually contracts by $1/7$, but the general envelope uses the floor and trace rather than this exact one-dimensional innovation spectrum. It also accounts for only one of two output packets being selected.

The bound holds for the mathematical exact trajectory under the specified hypotheses. It is not a timing bound, a claim about integer bit growth, or a hardware floating-point stability result. Equality of cached and recomputed gains transfers this already proved convergence statement to the cache algorithm without replacing the underlying gain sequence.

Source route: `GreedyPacketConvergence.lean`; `CachedPacketTrajectory.lean`; `PROOF_NOTE.md`.

16 Finite fuel is sufficient, not a forecast

The theorem supplies computable iteration fuel

$$N = \left\lceil \frac{a(B_0)}{(\varepsilon/2)^2(1-r)} \right\rceil_{\mathbb{N}}.$$

Its role is to guarantee a state with score at most $\varepsilon^2/4$. The search checks the current score before updating. At fuel zero it still checks the last permitted state, so the final bound can produce a successful return rather than being lost to an off-by-one branch.

The coordinate model has $a(B_0) = 34$, $r = 85/98$, and $\varepsilon = 1/20$. Its sufficient fuel is 410093, but its actual search stops after six updates. The rotated example has sufficient fuel 4267 and stops after four. These differences are expected: a uniform sufficient bound need not closely predict a small structured example.

The returned gain has an origin theorem. It is an actual greedy iterate B_k for some $k \leq N$, not an arbitrary candidate introduced after search. A separate score theorem proves that the returned state's exact score passes the search threshold.

Resource caps in the Python producer are an additional execution constraint. The ideal mathematical theorem does not imply that every problem finishes inside a fixed memory or time limit. A capped producer may report resource failure without contradicting finite termination of the mathematical program.

Source route: `CachedPacketSearch.lean`: `search_origin`, `search_score`, `search_complete`; `AdaptivePacketSolver.lean`.

17 Stopping slack protects the final certificate

The posterior theorem would permit the exact score condition $a(B) \leq \varepsilon^2$. The cached search deliberately aims for the stricter condition

$$a(B) \leq \frac{\varepsilon^2}{4}.$$

This gives the outward-rounded final verifier room to overestimate the score. The solver can then choose sufficient precision without requiring exact equality at the acceptance threshold.

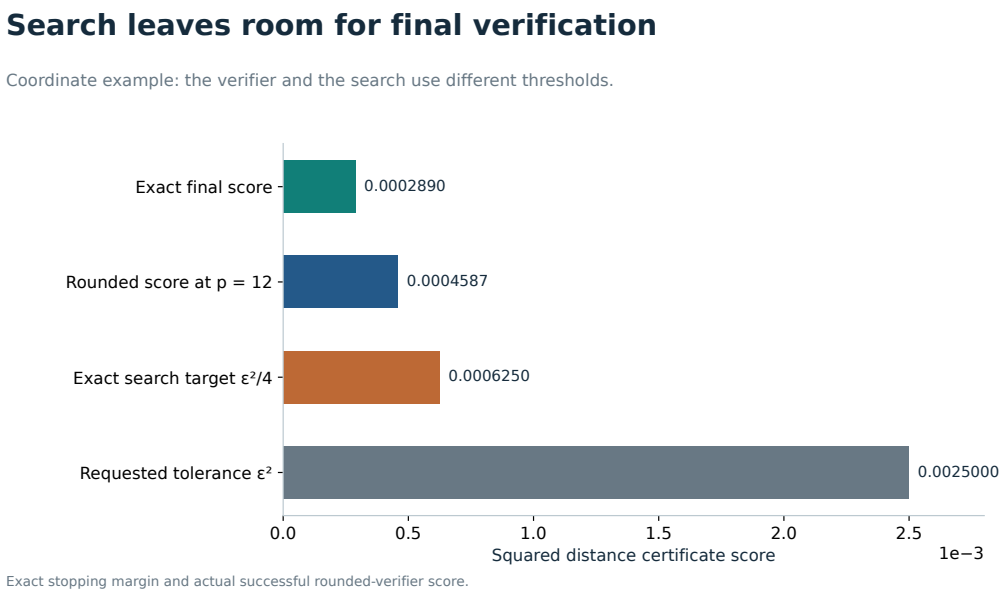


Figure 11: The search threshold reserves room inside the posterior tolerance. Passing exact search and passing final rounded verification are distinct events.

For the coordinate example, the search threshold is $1/1600$ and the final acceptance threshold is $1/400$. The final exact score $34/117649$ lies below the first. The accepted rounded score $481/1048576$ lies above the exact score but below the second.

This policy accounts for a change from the earlier adaptive checkpoint: its coordinate example could stop after five updates by attempting rounded verification sooner, while the cached release waits for the stricter exact-score margin and takes six. The recomputed selective benchmark uses the same stricter threshold, so its comparison with caching follows the same gain trajectory and candidate.

Slack does not make the final checker dispensable. It proves that a sufficient precision can succeed for a valid run. The actual candidate still passes the independent outward-rounded check before the solver returns the posterior guarantee.

Source route: `CachedPacketSearch.lean`; `CachedPacketSolver.lean`; `PrecisionCompleteness.lean`.

18 The final checker sees the returned gain

Final verification evaluates obligations from the model and the returned gain. It does not simply accept the cache's stored total as a proof. This distinction is what allows soundness to survive the rounded-cache counterexample later in the guide.

The evaluator uses a declared outward-rounded graph for model, residual, packet, and score quantities. Operationally, `certifyRounded` executes `checkSetup`, constructs `roundedCandidate`, and executes `checkTolerance`. Its soundness theorem then proves that the constructed candidate also passes `checkCandidate`. That latter exact candidate check is not an extra residual-recomputation step executed by the canonical final evaluator.

The squared tolerance is compared with the outward score. In the coordinate fixture the final candidate contains exact rational gain entries, packet upper bounds, and an innovation upper bound. These fields are evidence for the original exact model. A separate exact checker can recompute finite obligations for an untrusted supplied candidate; the source keeps that interface distinct from construction of a candidate by the canonical outward evaluator.

The soundness theorem for a successful return does not need a caller to assert that an arbitrary cache was valid. The independent final check is sufficient for the posterior implication. Termination is different: its proof uses valid initialization and cache preservation to guarantee that search and final precision selection succeed. Keeping soundness and termination separate prevents an invalid optimization from being defended merely because a checker remains sound.

Source route: `RoundedCandidate.lean`; `RoundedPosteriorSound.lean`; `CachedPacketSolver.lean`: `solveCached_sound`, `solveCached_terminates`.

19 A rejected precision is an honest part of the run

The coordinate execution tries precisions 4 and 12 for its final certificate. At precision 4, its rounded score is $25/16$, so the checker rejects it. At precision 12, the rounded score is $481/1048576$, and the checker accepts it against $\varepsilon^2 = 1/400$.

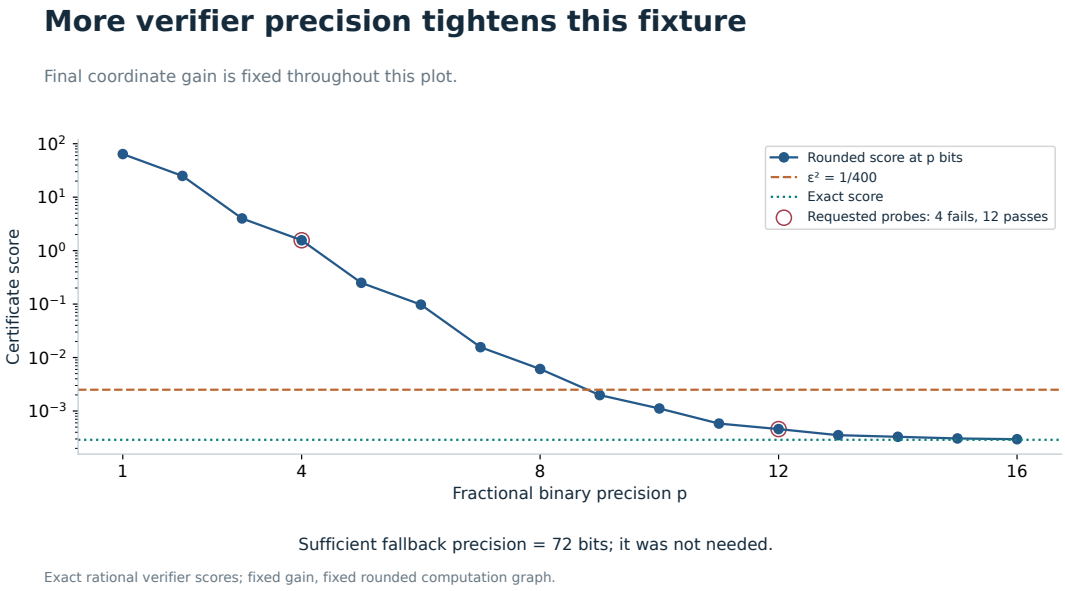


Figure 12: The coordinate gain's certificate curve at precisions 1 through 16 rejects probe 4 and accepts probe 12. The sufficient fallback precision is 72. This fixture does not assert strict monotonicity for every precision curve.

The rotated execution shows the same qualitative behavior with different exact numbers: $435/64$ at precision 4 and $9111/1048576$ at precision 12, compared with $\varepsilon^2 = 1/16$.

These dramatic coarse bounds arise from a particular evaluator and graph. They do not mean every quantity rounded to four fractional bits has that relative error. Error terms accumulate through the declared calculation, and conservative upper bounds need not be close at a very low precision.

The solver tries supplied precision probes and has a sufficient fallback. The precision-completeness argument uses exact-score slack and an explicit bound on rounded inflation. A failed early probe is preserved as evidence rather than omitted from the result. The guarantee attaches to the successful checked precision, not to a general assertion that the producer's first choice was adequate.

Source route: `PrecisionCompleteness.lean`; `TaggedPrecisionSearch.lean`; `results/cached_coordinate.json`; `results/cached_rotated.json`.

20 Dyadic intervals are enclosures, not rounded guesses

A dyadic grid at fractional precision p has spacing 2^{-p}. Downward and upward rounding choose grid endpoints that enclose an exact rational. A center-radius representation records an interval $[c-\rho, c+\rho]$ that contains the intended exact value.

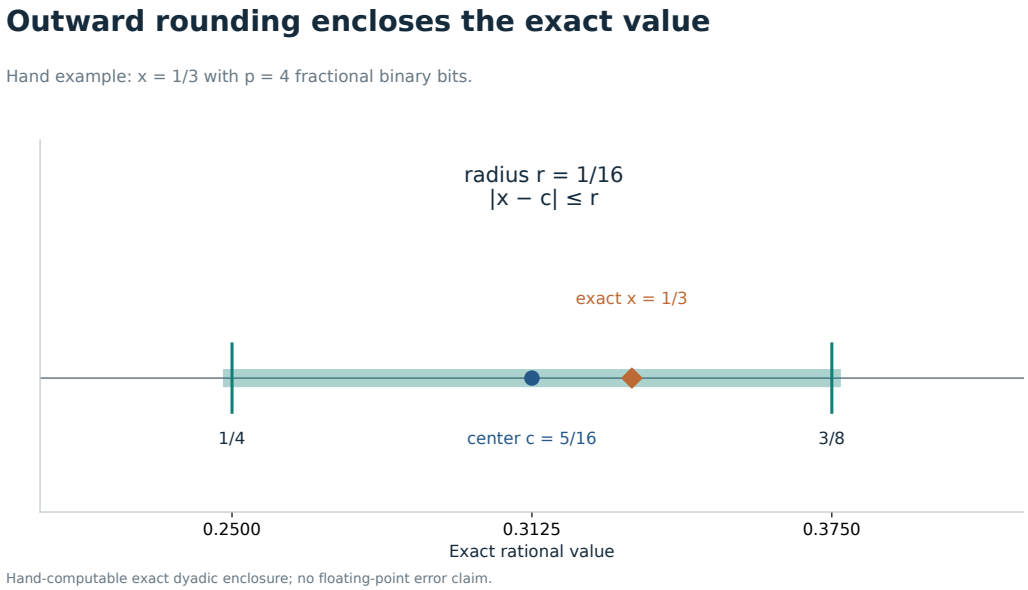


Figure 13: At four fractional bits, downward rounding of $1/3$ gives center $5/16$. Upward rounding of its lost magnitude gives radius $1/16$, so the ball $[1/4, 3/8]$ encloses $1/3$.

Addition and multiplication must propagate radii as well as centers. If an operation drops a radius, the stored center can still look plausible while the enclosure theorem is false. The retained negative controls include a rejection associated with dropping a rounding radius.

The arithmetic here is mathematically dyadic and explicitly modeled. It is not an automatic contract for processor floating point, fused multiply-add, overflow, underflow, or a library's matrix multiplication routine. Those behaviors require their own implementation refinement and error treatment.

For a fixed exact expression, increasing precision gives sufficient control of the error through the proved evaluator bounds. The solver uses this to produce a successful certificate and then a parameter representation with requested radii. Certificate precision and output precision can differ because the two calculations have different expressions and different acceptance conditions. In the worked examples these are 12 and 32 fractional bits.

Source route: `DyadicBall.lean`; `DyadicPrecision.lean`; `RationalBall.lean`; `results/NEGATIVE_CONTROLS.json`.

21 The returned intervals describe one named Gaussian

Once the gain is certified, the output evaluator represents its innovation mean m_B and Joseph covariance C_B. Every entry is returned as a center and nonnegative radius. The checked conditions are

$$|(m_B)_i - c_i| \le \rho_i \le \delta, \qquad |(C_B)_{ij} - c_{ij}| \le \rho_{ij} \le \delta.$$

The arrays and reported precision are tied to the actual evaluator output, rather than merely asserted to have the right shape.

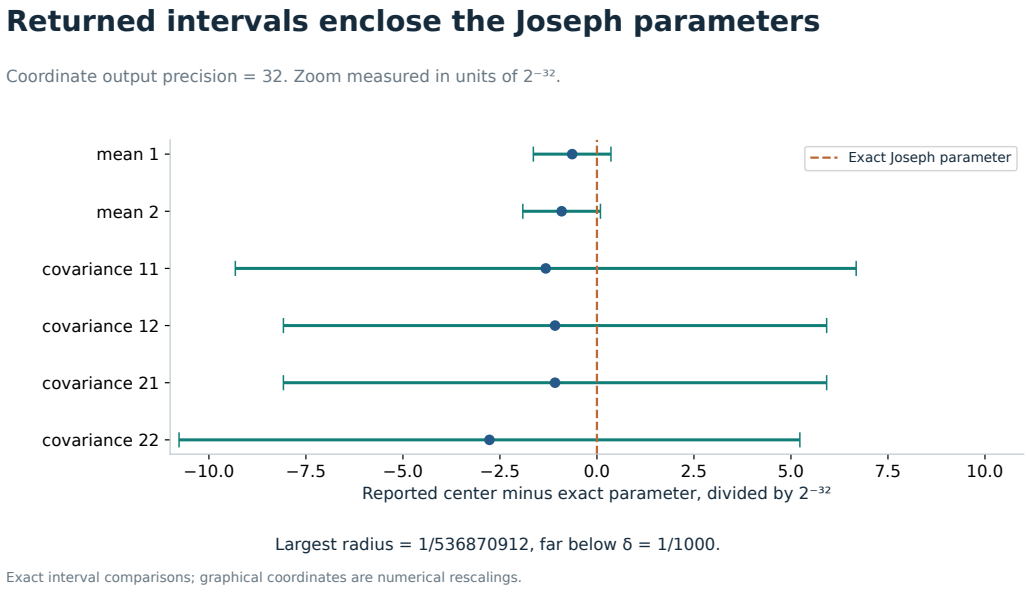


Figure 14: Intervals are translated relative to exact Joseph parameters and multiplied by 2^{32} for visibility. Each encloses the dashed zero line. They describe the approximation at B_6, and the centers alone do not define another certified Gaussian.

In the coordinate result, output precision is 32. Mean radii are $1/4294967296$, and the maximum covariance radius is $1/536870912$. These values are far smaller than the requested $\delta = 1/1000$. The rotated output has maximum radius $7/4294967296$.

The intervals remain useful even when a radius is visually invisible at the plot's natural scale. Exact numerical labels retain the quantitative information, and a magnified inset can explain the enclosure. A picture must never imply that a magnified interval has the width of the original uncertainty distribution.

Source route: `PosteriorOutput.lean`; `PosteriorOutputPrecision.lean`; `PosteriorOutputSemantics.lean`; `GeneratedCachedCoordinate.lean`.

22 Why epsilon and delta cannot be exchanged

The guarantee $W_2(\Pi_y, \Pi_{B,y}) \leq \varepsilon$ compares two probability laws. The output guarantee $\rho \leq \delta$ describes arithmetic enclosures of the approximation's own mean and covariance. One addresses inference approximation. The other addresses representation of that approximation.

For the coordinate case, reducing δ would ask the output evaluator to represent m_{B_6} and C_{B_6} more tightly. It would not change their exact difference from m_* and C_*. Conversely, reducing ε generally asks for more gain updates and a tighter posterior certificate, but it does not by itself define the radii of printed parameter intervals.

The covariance centers are not a separately certified covariance matrix. Entrywise enclosures do not imply that the matrix of centers is positive semidefinite, nor do they certify a Gaussian with center mean and center covariance. The release's represented law is identified by the exact returned gain and its Joseph formula, with intervals enclosing its parameters.

The Wasserstein theorem can support further consequences for expectations under suitable hypotheses, but such consequences require their own statements. This guide does not convert ε into arbitrary entrywise posterior parameter error or identify δ as uncertainty in the physical observation. The two tolerances are useful precisely because their meanings remain explicit.

Source route: `RELEASE_SCOPE.md`; `PosteriorOutputSemantics.lean`; `PosteriorEnclosureBounds.lean`.

23 Rotated packets organize a singular model

Return to the rank-one prior, with $H = I$, $R = I$, $m = 0$, and $y = (1, -1)^{\mathsf{T}}$. Use the rational projectors

$$Q_0 = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \qquad Q_1 = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}.$$

They project onto the diagonal and anti-diagonal directions. They are complete, orthogonal, and self-adjoint. The input packet family is the same pair for this fixture.

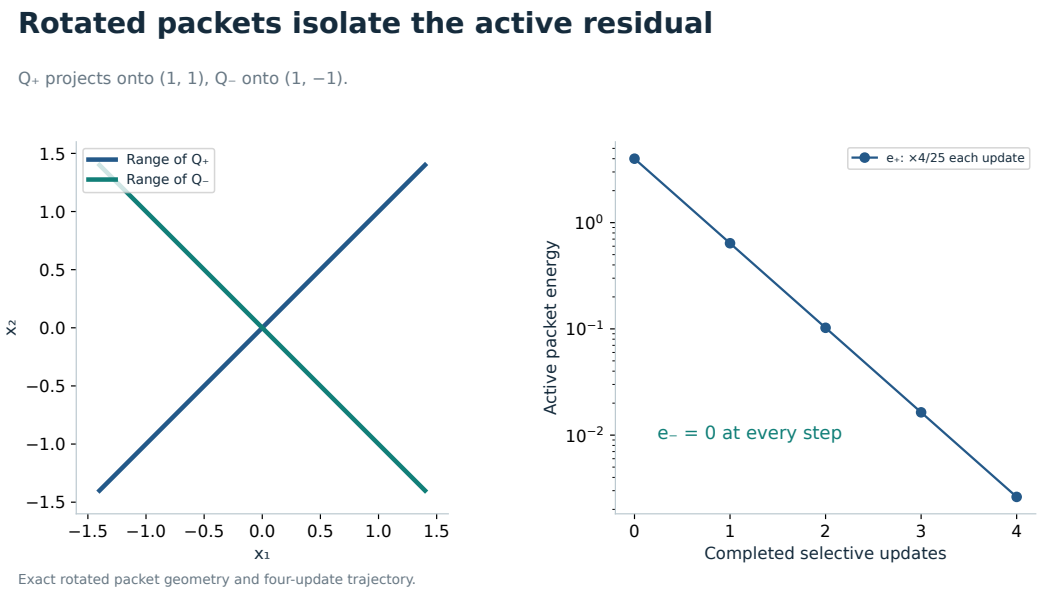


Figure 15: The diagonal packet contains all prior residual energy. The anti-diagonal packet contains none, even though the observation lies in that direction.

Initially $F(B_0) = P$, so $Q_0 F(B_0) = P$ and $Q_1 F(B_0) = 0$. The residual energy is 4, and $w = 3$. Hence the initial score is 12.

The example separates three ideas that can otherwise be conflated: the state support, the direction of the supplied observation, and the output packet carrying gain-equation residual. Selection is based on the last of these. The nonzero observation does not force an update of the packet aligned with it. The residual is a matrix determined by H, P, R, B, while observation magnitude enters its scalar posterior-error weight.

Source route: `examples/adaptive_rotated.json`; `GeneratedCachedRotated.lean`; `RationalPacketCertificates.lean`.

24 Four rotated updates and their final law

Here $\tau = 1/5$ and the selected diagonal residual contracts by $2/5$. Every step selects packet 0, giving total energies

$$4,\quad \frac{16}{25},\quad \frac{64}{625},\quad \frac{256}{15625},\quad \frac{1024}{390625}.$$

The final gain is

$$B_4 = \frac{203}{625}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix},$$

and the exact score is $3072/390625$. It passes $\varepsilon^2/4 = 1/64$ for $\varepsilon = 1/4$. At three updates the score $768/15625$ does not pass this stricter condition.

The exact gain is $P/3$, and both exact and approximate means are zero because their gains annihilate $y = (1, -1)^{\mathsf{T}}$. The exact covariance is $P/3$. The final Joseph covariance is

$$C_{B_4} = \frac{130379}{390625}P = \left(\frac{1}{3} + \frac{512}{1171875}\right)P.$$

It remains rank one, so both laws are supported on the same diagonal line.

This example demonstrates singular-prior support and genuinely rotated rational packets. It does not establish that such packets are always a better decomposition. The packet choice is fixed input data. Comparing or adapting decompositions would require a separate criterion and theorem relating the resulting computations.

Source route: `results/cached_rotated.json`; `GeneratedCachedRotated.lean`; `FDInexactGainAlgebra.lean`.

25 A rounded cache can announce false convergence

Consider the scalar model $P = H = R = 1$, $m = 0$, $y = 1$, and $B_0 = 0$. Its innovation is $S = 2$, its step is $\tau = 1/3$, and its weight is $w = 2$. One exact update gives

$$B_1 = \frac{1}{3}, \qquad F(B_1) = \frac{1}{3}.$$

Now independently round both gain and cached residual downward to zero fractional bits. The stored gain becomes 0 and the stored residual becomes 0.

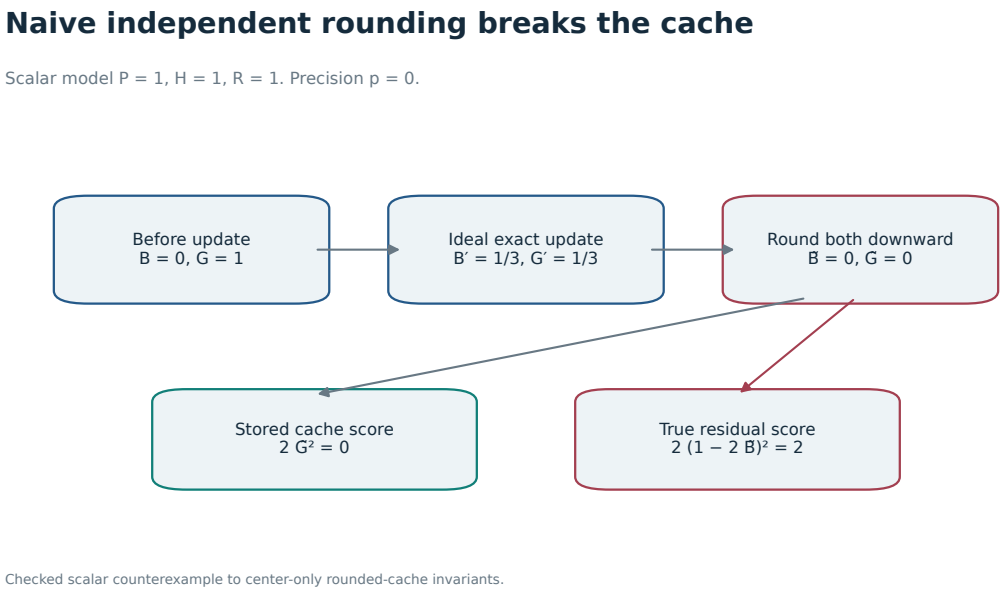


Figure 16: Independent zero-bit rounding produces a stored score of zero at a gain whose true residual score is two. The exact cache invariant has been destroyed.

The cache reports energy zero. Yet the true residual at its stored gain is $F(0) = 1$, so the true score is 2. The mismatch is not a small relative error near a solution. It is a false convergence signal introduced by forgetting the relationship between rounded gain and rounded residual.

The source formally proves the stored zero score, the true score of two, and failure of cache validity. This refutes the naive center-only rounded route. It does not refute the possibility of a rounded selective solver that maintains certified error enclosures. Such a solver needs a new invariant relating stored quantities to the actual gain residual, rather than reusing the exact invariant after rounding.

Source route: `CachedRoundingBoundary.lean`: `naive_score_zero`, `true_score_two`, `naive_cache_invalid`.

26 The independent verifier still rejects the bad gain

For the scalar counterexample, request $\varepsilon = 1/10$. The final independent verifier rejects the stored gain $B = 0$. The checked theorem is stronger than observing that a diagnostic residual reconstruction noticed a problem: the actual certificate routine returns no accepted candidate for the stated test.

This establishes a useful separation. Soundness of accepted outputs survives an incorrect internal convergence signal, because acceptance is independently checked from the model and gain. The naive optimization nevertheless loses its promised termination argument. It may stop at an internally reported score zero and then fail final verification, even though its intended contract promised successful return for valid data and positive tolerance.

A future repair must budget at least the discrepancy between the cached residual representation and $F(B)$, propagate that discrepancy through selected updates, and account for selection using uncertain energies. It must also show that final precision and iteration eventually suffice. Naming those requirements does not supply their proof.

The release takes the exact cached branch to completion and preserves the failed rounded route as a checked boundary example. This is more informative than presenting the final checker as a general cure for every implementation change. A checker establishes the implication attached to an accepted candidate. It does not automatically make a modified search complete, efficient, or faithful to the original gain trajectory.

Source route: `CachedRoundingBoundary.lean`: `naive_final_verification_rejected`; `CachedPacketSolver.lean`; `FAILURE_LOG.md`.

27 The arithmetic boundary is deliberately narrow

The canonical cached branch has exact rational model data, gains, residual matrices, energies, comparisons, and updates. Its final certificate and parameter output use explicit outward-rounded dyadic operations. These arithmetic regimes are connected by proofs about the declared operations.

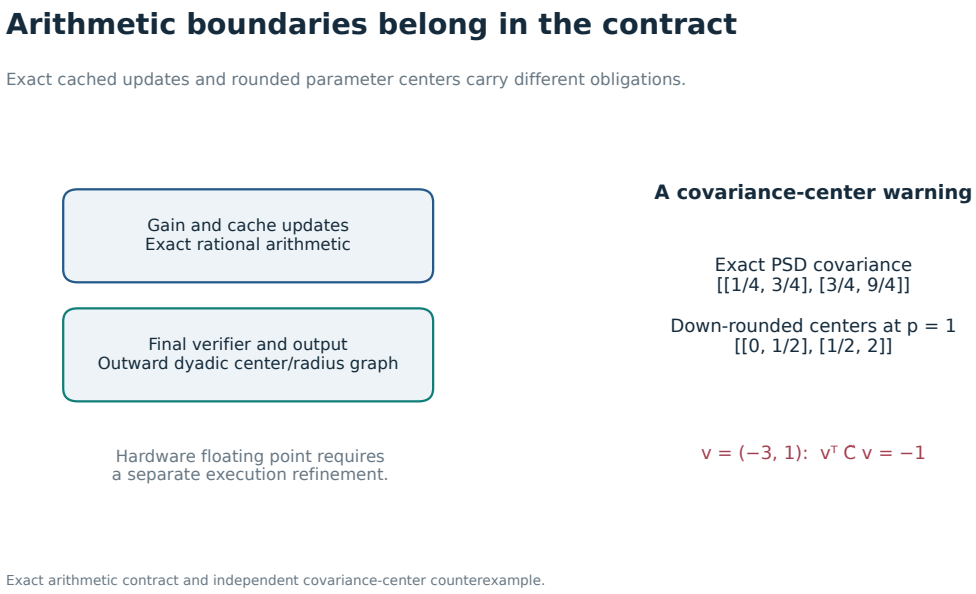


Figure 17: Exact iteration and outward-rounded evaluation have separate roles. A rank-one covariance rounded to centers alone has quadratic witness -1. This refutes center-only covariance interpretation without invalidating enclosing arrays.

For the plotted center example, the exact covariance $\left(\begin{smallmatrix} 1/4 & 3/4 \\ 3/4 & 9/4 \end{smallmatrix}\right)$ is positive semidefinite. Downward half-grid centers give $\left(\begin{smallmatrix} 0 & 1/2 \\ 1/2 & 2 \end{smallmatrix}\right)$, whose quadratic form at $(-3,1)^{\mathsf{T}}$ is -1. This is a fresh exact rational diagnostic, not a new Lean declaration.

The source producer rejects raw JSON floating-point numbers. Integer values, fraction strings, and decimal strings can be decoded as exact rational data. This keeps the emitted model precise, but does not prove universal parser refinement.

Likewise, the reference Python uses rational arithmetic and is useful for producing and inspecting examples. A proof-producing instance certifies the explicit rational model and program result in emitted Lean. It does not certify arbitrary changes to Python parsing, arbitrary native matrix libraries, or timing measurements merely because those changes printed the same status label.

The mathematical sampler construction used in the Wasserstein coupling is also distinct from an executable random sampler. The theorem uses actual Gaussian laws and their affine maps. The release provides no runtime sampler contract, distributional test certificate, or hardware random-number-generator guarantee.

Source route: `README.md`; `RELEASE_SCOPE.md`; `DyadicBall.lean`; `FDGainSampler.lean`.

28 Three solver branches should stay distinguishable

The source preserves a rounded full-gain solver, an adaptive recomputed selective solver, and the canonical exact cached selective solver. They share posterior and output semantics but have different iterative arithmetic and stopping policies.

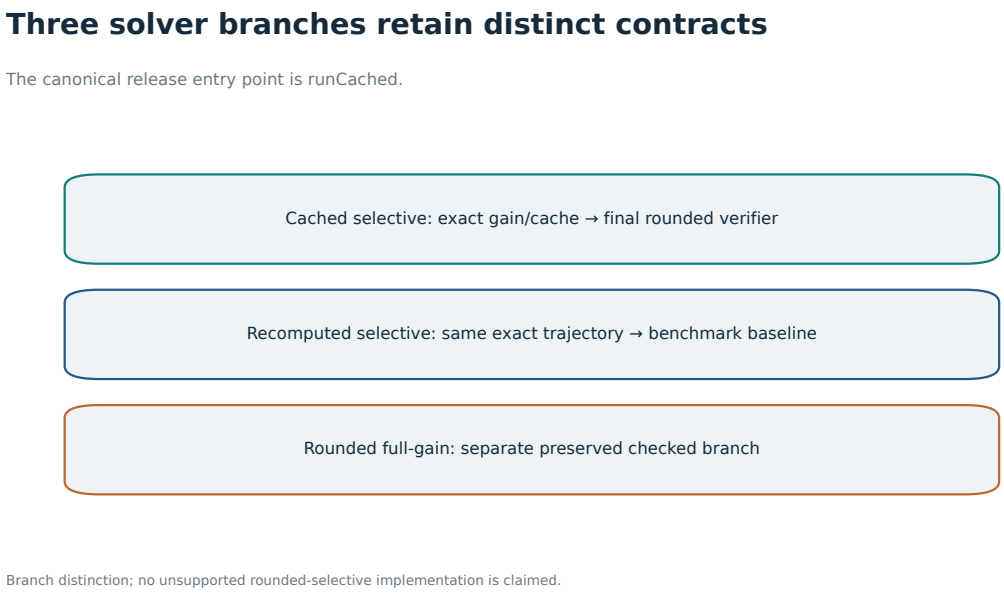


Figure 18: The preserved branches share a target law and certificate meaning, while their iteration and search definitions differ. The exact cache theorem belongs to its specified branch.

The rounded full-gain branch genuinely rounds its updates and has its own termination theorem. The adaptive selective branch recomputes packet information and can attempt a rounded certificate earlier. The cached selective branch updates exact stored residuals and waits for the stricter exact-score margin before final certification.

Equality of cached and recomputed greedy gain trajectories is a statement about corresponding exact greedy steps with the same selection convention. It is not a claim that every branch returns the same gain after its own stopping policy. The controlled benchmark therefore aligns the cached and recomputed selective stopping thresholds before comparing their outputs.

The failed naive rounded cache should not be confused with the retained rounded full-gain solver. Their definitions and invariants differ. A negative result for one center-only cache rule does not invalidate a separately proved rounded solver. Conversely, the rounded full-gain proof does not establish the missing rounded-selective cache invariant.

Source route: `RoundedSolverSearch.lean`; `AdaptivePacketSolver.lean`; `CachedPacketSolver.lean`; `PROOF_NOTE.md`.

29 A proof-producing instance names its actual result

The coordinate and rotated instances do more than submit final gains. Their generated Lean modules check the fuel, selected labels, exact and rounded scores, actual solver return, output precision, returned centers and radii, and the final result-validity proposition.

For a new input the reference producer can write an emitted Lean instance. The producer's status is explicitly provisional until that instance compiles. This labels the boundary honestly: rational execution can produce candidate data, while elaboration and kernel checking establish the declared proof for those explicit data.

The universal theorem supplies the contract for the mathematical program under its hypotheses. A concrete compiled instance ties a particular model, requested tolerances, gain, and output arrays to that program. These are related forms of evidence with different jobs. A table of timings is not a substitute for either.

In the archived benchmark, the 21 exported modules are final-candidate posterior certificates. They do not certify every measured Python iteration trace or every timed output array. The two worked full-program instances provide the stronger execution/output tie for their own examples. This distinction lets the publication present useful benchmark evidence without upgrading static candidate certificates into a universal implementation-refinement claim.

Source route: `GeneratedCachedCoordinate.lean`; `GeneratedCachedRotated.lean`; `results/FORMAL_INSTANCES.json`; `checks/cached_posterior.py`.

30 Valid input and available resources are different

The mathematical theorem assumes exact factor-coded model data, a positive noise floor, fixed valid packet families, a covering finite scan, and positive rational ε and δ. Under those hypotheses the ideal program has sufficient finite iteration, verification precision, and output precision.

The reference producer imposes explicit practical caps. Its defaults include dimensions at most eight, at most 10000 updates, rational numerators and denominators bounded by a bit budget, finite precision probes, and limits on scan length and input size. These limits make a reference workflow inspectable and prevent an execution from being presented as success after exhausting its resources.

A resource-limited run can fail while the mathematical input remains valid. Conversely, a small file can be invalid because its packets fail completeness or its tolerance is nonpositive. A clear implementation distinguishes these outcomes rather than combining them into a generic convergence claim.

The release also does not prove a universal theorem that arbitrary JSON files denote the same model as a Python execution. The generated Lean source contains the explicit rational data that are certified. A future parser-refinement theorem would connect an accepted serialized input to that model representation. Until then, inspectable emitted data and precise provisional status are part of the workflow's evidence boundary.

Source route: `README.md`; `checks/cached_posterior.py`; `CachedPacketOutput.lean`; `RELEASE_SCOPE.md`.

31 The cache saves iteration work, not an overall victory

The controlled seven-case benchmark compares cached selection, recomputed selection, and an efficient full Richardson baseline. All use exact rational arithmetic, the same tolerances, the same stricter score margin, and a common final certificate/output policy. Cached and recomputed selective runs return identical gains.

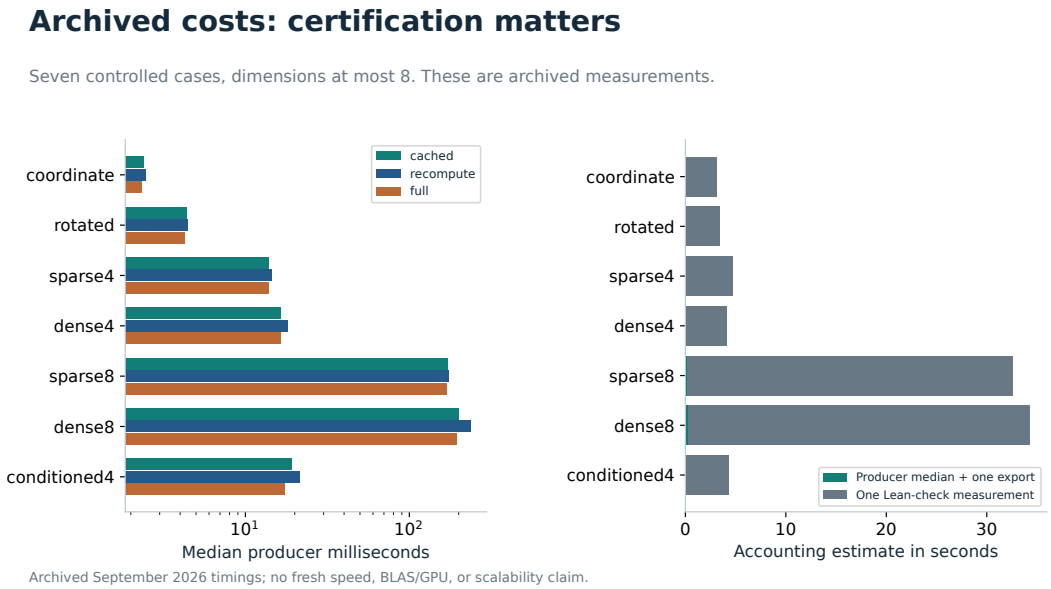


Figure 19: Archived bounded timings separate producer cost from export and Lean checking. Cached iteration is cheaper than recomputed selection, while total costs do not establish superiority over the full-update baseline.

The coordinate case takes six selective updates and three full updates. Its archived median producer times are 2.38 ms cached, 2.46 ms recomputed, and 2.33 ms full. In the dense eight-dimensional case the respective times are 200.69, 235.15, and 195.44 ms. These measurements show why local savings and total cost must be distinguished.

One warmup precedes five measured repetitions with rotating method order. The benchmark covers dimensions at most eight and includes dense residuals, concentrated residuals, a singular prior, rectangular observations, and a smaller noise floor. Final verification, parameter output, export, and Lean checking add substantial cost at this range.

No practical superiority over optimized numerical packages follows. The iterative counters omit preprocessing, certificate arithmetic, integer bit growth, and allocation. They are diagnostics of a reference implementation, not a formal bit-complexity theorem or a scalability measurement.

Source route: `BENCHMARK_REPORT.md`; `results/FULL_COST_BENCHMARK.json`; `computations/release-final/benchmark/benchmark.json`.

32 What the verification record establishes

The checked source release records a clean build of all 117 active Lean modules and an axiom audit covering 543 theorem and lemma declarations. Its allowlist is `propext`, `Classical.choice`, and `Quot.sound`. These counts include support statements and benchmark certificates, rather than counting independent mathematical discoveries.

The receipt binds sources, outputs, logs, and dependencies to the recorded build. The portable runtime is separate from the source archive, with Lean and mathlib pins specified for reproduction. A fresh source hash comparison can confirm that archived source files have not changed. A fresh Python check can confirm the tested exact arithmetic examples. Neither fact by itself amounts to a new Lean kernel run.

Negative controls are part of the evidence. The archive preserves rejection of a placeholder, a false equality, and a dropped-rounding-radius attempt. The scalar rounded-cache counterexample is a positive checked theorem about a failing proposed optimization. These two forms of negative evidence have different meanings, and both help prevent a permissive or misunderstood workflow from being mistaken for proof.

This paper set reports archived verification and current checks separately. If the pinned runtime is unavailable, a new compilation cannot be claimed. The source, reproduction instructions, archived evidence, exact examples, and limits remain available for an independently reproducible verification.

Source route: `FINAL_VERIFICATION_RECEIPT.json`; `VERIFICATION_SUMMARY.md`; `results/NEGATIVE_CONTROLS.json`; `MODULE_ORDER.json`.

33 The assurance map separates theorem from interface

The strongest completed claim concerns the mathematical program and the explicitly represented Gaussian law. Under the declared hypotheses, the canonical cached program terminates successfully and returns the posterior tolerance and parameter enclosures described in the contract.

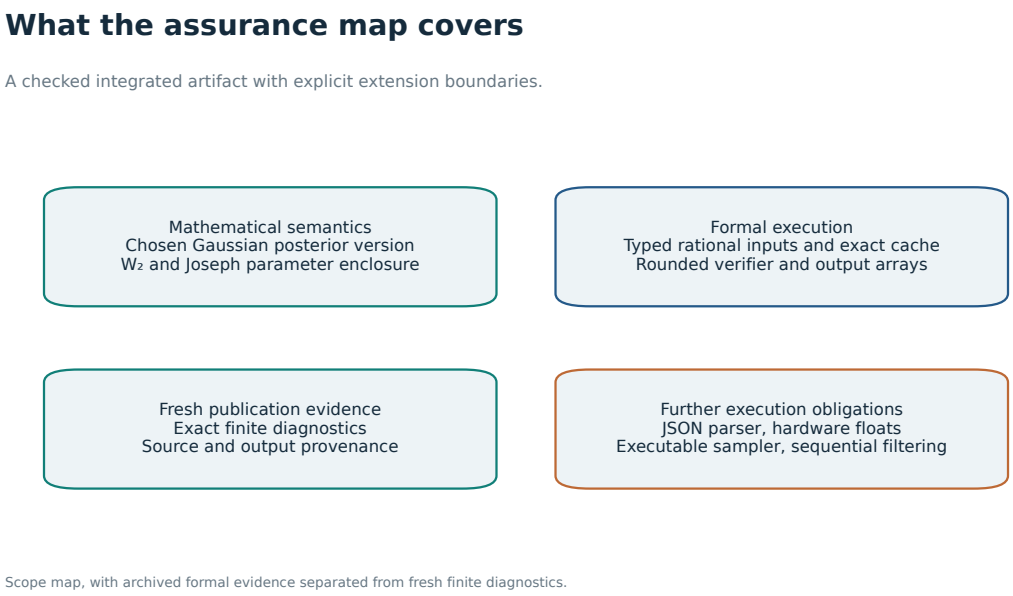


Figure 20: The assurance map records completed theorem obligations and the remaining boundaries around parsing, hardware arithmetic, random sampling, and broader inference workflows.

The exact cache is established by initialization and preservation, rather than supplied as an assumption by a caller. The gain is tied to an actual greedy iterate. Final soundness uses independent verification. Output intervals refer to the same Gaussian as the posterior theorem. Those are completed links within the scope of the release.

Outside that scope lie universal Python refinement, serialized-input refinement, hardware floating point, an executable sampler contract, and sequential filtering. Keeping those boundaries explicit does not make the declared theorem conditional on unfinished work. It prevents the finished theorem from being described as a larger software or statistical guarantee than it provides.

Literature comparisons identify established ideas behind Gaussian conditioning, greedy block selection, residual caching, coupling bounds, and formal arithmetic. The publication's demonstrated contribution is their checked integrated artifact. The guide does not infer broad novelty or priority from the number of modules or the completeness of the release obligations.

Source route: `SELF_REVIEW.md`; `RELEASE_SCOPE.md`; `LITERATURE_COMPARISON.md`; `CachedPacketOutput.lean`.

34 Extensions have concrete proof obligations

The rounded selective extension needs a cache-error invariant, certified energy comparisons, and a perturbation argument that still yields sufficient convergence and final-check slack. The scalar counterexample rules out dropping those obligations.

A changing hierarchy would need a relation between the old and new packet decompositions, a rule for transferring residual information, and a proof that completeness and error accounting survive refinement. Sequential filtering would need to connect one returned posterior representation to the next model and track accumulated inference and representation error. Neither follows merely by placing the one-update function in a loop.

Uncertain inputs would require enclosures or a probability model for H, L, N, m, y and a theorem relating the family of possible exact models to a returned result. A hardware implementation would require a precise arithmetic/refinement contract. A sampler would need a runtime distributional semantics and a proof connecting its actual law to the represented Gaussian.

Performance work has another set of obligations. Larger structured cases, final-verifier cost, integer bit growth, and proof-checking cost need separate measurement or complexity analysis. The bounded benchmark does not choose a winner for those unexplored settings.

These are research directions with defined missing links. They do not reopen the exact cached one-observation contract, which is the finished core documented here. They do identify where a future release must add evidence before using a stronger name for its capabilities.

Source route: `RELEASE_SCOPE.md`; `ROUTE_RECORD.md`; `FAILURE_LOG.md`; `BENCHMARK_REPORT.md`.

35 Reading the complete result back from its evidence

Begin with the model factors and the observation. They determine P, R, S, v, the target posterior version, and the residual weight. The fixed packet families then determine how unresolved gain error is divided. Initialization constructs an exact cache, and every update preserves its connection to the true residual.

The greedy scan supplies a specified selected label. Cached and recomputed gains agree along the corresponding exact trajectory. The geometric bound and computable fuel guarantee that the stricter exact-score threshold is reached. An origin theorem identifies the gain actually returned from search.

Final outward-rounded evaluation verifies the candidate independently. The posterior theorem converts that accepted residual bound into a Wasserstein guarantee against the explicit Gaussian posterior version. A separate precision calculation returns intervals around the mean and covariance of the Joseph Gaussian at that very same gain.

The coordinate example makes these links concrete through six updates. The rotated example shows that a singular prior and noncoordinate packets fit the contract. The naive rounded-cache example explains why changing arithmetic requires a new invariant. The benchmark explains why a correct optimization need not yield an overall speed advantage.

The result is a complete, scoped inference artifact: a mathematically specified solver, a termination argument, posterior semantics, arithmetic certificates, parameter enclosures, inspectable examples, and preserved verification evidence. Each item can be read alongside the traditional paper and the Lean companion, with reproducible exact figures anchoring the route.

Source route: `PROOF_NOTE.md`; `CachedPacketOutput.lean`; `GeneratedCachedCoordinate.lean`; `GeneratedCachedRotated.lean`.